PURESCRIPT

Trends

- The New Renaissance
- Functional programming (re)gains popularity
- Types gain popularity in functional programming
- Complexity is growing
- Legacy code is growing
- Javascript is ubiquitous

Problems

- As applications complexity grows, correctness and evolvability suffer
- As legacy codebase grows, refactoring & maintenance becomes very hard
- Existing tools:
    - too low level
    - provide wrong or insufficient abstractions:
        - imperative/mutable
        - non-composable
        - cannot be proven correct
    - don't use proper (logical) design tools

Requirements

- Modularity
- Abstraction
- High level
- Proper design tools
- Tooling
- Ecosystem
- Refactorability/maintainability
- Safety

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[dr, "g \circ f"'] & Y \arrow[d, "g"] \\
 & Z
\end{tikzcd}$$

Use cases

- End-to-end Javascript application development
- Functional application logic, JS interface to the real world via FFI
- Server-side or client-side Javascript

- Other backends (C++, Lua, etc)

Solutions

- ES6
  - still js, too low level, no design, no constraints
- Dart
  - not js, too low level, good target language probably
- Compiled-to-js languages
  - dynamic/unityped - not good enough, even ClojureScript
  - static - type systems not good enough, even Typescript, Flow
  - too complex and heavy - GHCJS, Fay, Haste, Scala.js, Funscript
  - too specialized - Elm
  - right one - ?

PURESCRIPT

PURESCRIPT is a small strongly typed programming language that compiles to JavaScript.

Portrait of Young Woman with Unicorn, Raphael

Purescript *in tōtō*, part 1

- Powerful
  - types & type inference (H&M)
  - enables abstraction
  - if it compiles, it works - no browser “live reloads” or console debugging :-)
- Compact
  - no runtime
  - small, features implemented in libraries
  - (very) fine grained
- Compatible
  - leverages existing js tools
  - works with existing legacy js code
  - CommonJS compatible
- Flexible
  - simple FFI
  - can be used for parts of application or tests only
- Simple

Purescript *in tōtō*, part 2

- Batteries not included (Elm as a library)
  - virtual-dom
    - React, Thermite
    - Halogen - typesafe UI
  - reactive
    - signals, RxJS wrapper
- (co-, free-)monads to the rescue
  - composable effects
  - async as implementation detail
  - easy DSLs

- Multiple backends
  - browsers
  - Node
  - native (C++11 backend)
  - iOS and Android (React Native),
  - AWS Lambda
- Evolves fast
- Libraries evolve even faster

Purescript *in tōtō*, part 3

- Good learning resources
- Javascript object syntax
- Human readable output
- Fast parallel builds

- Is being used in production
- Active community
- Very productive community
- `.purs` file extension

Language Features

- Type Inference
- Higher Kinded Polymorphism
- Support for basic Javascript types
- Extensible records
- Extensible effects
- Optimizer rules for generation of efficient Javascript
- Pattern matching
- Simple FFI
- Modules
- Rank N Types
- Do Notation
- Tail-call elimination
- Type Classes

Language Features

- 类型推断
- 高阶多态
- 支持 JavaScript 基础类型
- 可扩展的记录
- 可扩展的副作用
- 生成高效的 JavaScript 代码的优化方案
- 模式匹配
- 简单的外部函数调用接口
- 模块
- Rank N 类型
- do 表达式
- 尾递归优化
- 类型类

Contrib

- signals - FRP as a library
- Thermite - React bindings
- Halogen - typesafe UI, better than React
- QuickCheck
- StrongCheck
- Pursuit search engine

Haskell?!

- Written in Haskell
- Similar to Haskell, but
  - simple Haskell
  - strict semantics
  - Javascript object notation

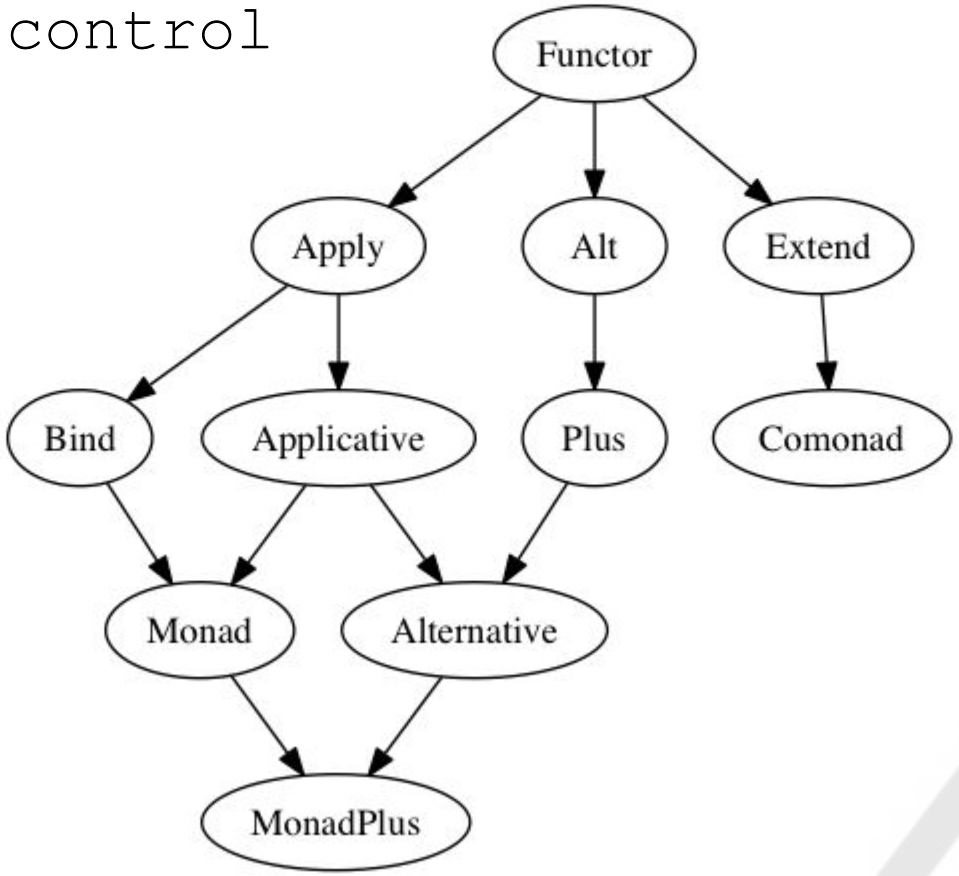
- Few improvements
  - typeclasses hierarchy
  - granular effects
  - explicit imports - no default Prelude etc
  - no legacy compatibility requirements
    - `import qualified`
    - `Unit, Array, a:as`
    - single `String` type

Differences from Haskell

- Explicit universal quantifier
- No Prelude imported by default
- `[a]` vs `Array a`, `()` vs `Unit` etc
- Granular effects - `IO` vs `Eff`
- Records with row types - js-compatible, with js-syntax
- Typeclasses syntax - <=, explicit names for instances
- No automatic instances deriving (yet)
- Type class hierarchies
- No built-in tuples
- Composition operator `(.)` vs `(<<<), (>>>)`
- No array comprehensions - use do-notation
- No special treatment for `$`
- No infix defining of operators (yet)
- No extensions, some built-in:
  - EmptyDataDecls
  - ExplicitForAll
  - FlexibleContexts
  - FlexibleInstances
  - MultiParameterTypeClasses
  - PartialTypeSignatures
  - RankNTypes
  - ScopedTypeVariables
- More generic functions - `Data.List` vs `Data.Foldable`, `Data.Traversable`
- Explicit type class exporting
- No cons `(a:as)`

Class hierarchy

`purescript-control`



Example 0

```
module Main where

import Prelude
import Control.Monad.Eff.Console (log)

main = do
  log "Hello, World!"
  main
```

Example 1

```
module Main where

import Prelude
import Control.Monad.Eff (Eff())
import Control.Monad.Eff.Console (log, CONSOLE())

repeat :: forall e. Int -> Eff e Unit -> Eff e Unit
repeat 0 _            = pure unit
repeat count action = do
  action
  repeat (count - 1) action

main :: Eff (console :: CONSOLE) Unit
main = repeat 2 $ log "Hello, World!"
```

Example 2

```
module Main where

import Prelude
import Control.Monad.Eff         (Eff())
import Control.Monad.Eff.Console (print, CONSOLE())
import Control.Monad.Eff.Random  (randomInt, RANDOM())

repeat :: forall e. Int -> Eff e Unit -> Eff e Unit
repeat 0 _            = pure unit
repeat count action = do
  action
  repeat (count - 1) action

main :: Eff (console :: CONSOLE, random :: RANDOM) Unit
main = repeat 2 $ randomInt 1 10 >>= print
```

Workflow

```
$ mkdir MyProject
$ cd MyProject
$ pulp init
$ edit src/Main.purs
$ pulp build
$ pulp run
```

Tooling

- Editor support
  - Atom
  - Sublime
  - IntelliJ
  - Vim
  - Emacs
- Docker, nix, npm, stack/cabal, homebrew, chocolatey, binaries

- node based - `bower, grunt, gulp, npm`
- Without node - `psc`, `git`, `psc-bundle`
- Documentation generation
- `pulp`
- `psc-ide`
- `psvm`
- REPL - `psci`
- Pursuit - like Hoogle

Learning resources

- Purescript book
- Github wiki
- purescript.org
- #purescript on Freenode
- Try Purescript
- github.com/purescript
- Intro to Purescript
- Async Purescript
- Better know a language: PureScript video
- Better know a language: PureScript slides
- Elm vs Purescript I-IV
- 24 Days of PureScript
- functorial.com
- twitter.com/purescript

Live demo

- twic https://github.com/EugeneN/twic
- pureGoL http://eugenen.github.io/pureGoL/
- Try Purescript

Eugene
Naumenko
eugene@traversable.one